Mobile SDK for Xamarin - SSC 1.1

All the official Sitecore documentation.

sitecore®
Own the experience™

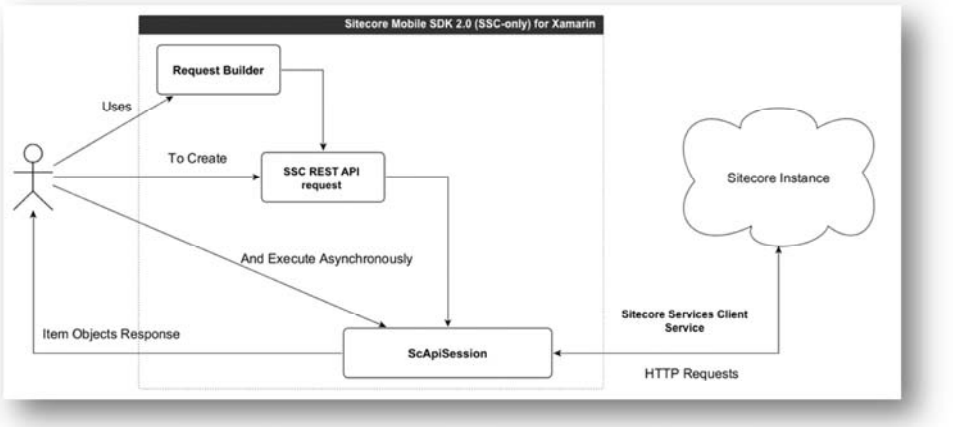
Accessing Sitecore content

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

The Sitecore Mobile SDK serves as an interface that connects the Sitecore Services Client REST service (SSC) with an application that lets users work with Sitecore items and their fields. Although the Sitecore Mobile SDK uses a RESTful request-response model, it lets developers work at a higher level of abstraction than HTTP requests and JSON responses.

To retrieve Sitecore content, build an SSC request object, send the request using the session object, and wait for the response.

The diagram represents the process of accessing Sitecore content using the SSC SDK:



To process requests asynchronously, the Sitecore Mobile SDK uses the `async` and `await` features of the .NET 4.5 Runtime. This enables the processing of all the requests using the same operation flow, which consists of the following steps:

- Authenticating
- Constructing an HTTP request
- Networking
- Parsing the response data

The Sitecore Mobile SDK does not cache the content downloaded from a Sitecore instance. Implement persistence logic for your application yourself.

Send feedback about the documentation to docsite@sitecore.net.

Building item requests

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

When you have configured a request with all the necessary parameters, you can build the request. Invoke the `Build()` method to get the request object.

```
var request =
ItemSSCRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
  .Database("master")
  .Language("fr")
  .AddFieldsToRead("Name")
  .AddFieldsToRead("Photo")
     .IcludeStanderdTemplateFields(true)
```

```
  .Build();
```

When you build a request, you must observe the following rules:

- Define the item identification parameter first. Define optional parameters in any order.

  Correct requests:

  ```
  ItemSSCRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
    .Database("master")
    .Language("fr")
    .Build();
  ```

  Or:

  ```
  ItemSSCRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
    .Language("fr")
    .Database("master")
    .Build();
  ```

  An incorrect request:

  ```
  var request = ItemSSCRequestBuilder.Database("master")
    .ReadItemsRequestWithPath("/sitecore/content/home")
    .Build();
  // This will not compile
  ```

- The `Add…` methods accumulate values, so consider the invocation order.

  For example, the following request retrieves the *Home* item and its two fields: the `Name` field and the `Photo` field.

  ```
  ItemSSCRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
    .AddFieldsToRead("Name")
    .AddFieldsToRead("Photo")
    .Build();
  ```

- Invoke methods that don't accumulate values only once.

  A correct request:

  ```
  ItemSSCRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
    .Database("web")
    .Language("en")
    .Build();
  ```

  An incorrect request that throws *InvalidOperationException*:

  ```
  ItemSSCRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
    .Database("master")
    .Language("fr")
    .Database("web")
    .Language("en")
    .Build();
  ```

Executing requests

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

When you request Sitecore items, all the requests are executed asynchronously by the *ISitecoreSSCSession* interface instance. The SDK contains the corresponding class, but it should not be instantiated directly.

ISitecoreSSCSession provides two overloads for the `ReadItemAsync()` method to process the requests based on one of the item identification parameters - *IReadItemsByIdRequest* and *IReadItemsByPathRequest*. For example:

```
ScItemsResponse response = await session.ReadItemAsync(request);
```

If you cannot use the `async` keyword with your method, use the `Task.WaitAll()` method instead:

```
var readHomeItemTask = session.ReadItemAsync(request);

Task.WaitAll(readHomeItemTask);

ScItemsResponse items = readHomeItemTask.Result;
```

To create, edit or remove items, use the following session methods:

- `CreateItemAsync` – for creating items.
- `UpdateItemAsync` – for editing items.
- `RunSitecoreSearchAsync` – for searching items.
- `DeleteItemAsync` – for removing items.

Send feedback about the documentation to docsite@sitecore.net.

Identifying Sitecore items

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

You can use the Sitecore Mobile SDKto create, retrieve, delete, update and search items in the Sitecore instance. To request a specific item, you should identify it by one of the following parameters:

- Item ID
- Item Path

Search request

You submit the item identification parameters as string objects, so you can submit any values. However, to be correctly processed by the Sitecore Services Client REST service, they must conform to a set of validation rules.

When these item identifications parameters are not sufficient to identify the item precisely, you can specify the item source.

The Sitecore Mobile SDKdoes not let you request an item by more than one parameter at a time, therefore, there are two separate request objects for retrieving items:

- IReadItemsByIdRequest
- IReadItemsByPathRequest

All requests are immutable read-only interfaces. To generate these requests, use the `ItemSSCRequestBuilder` class:

```
var request = ItemSSCRequestBuilder.ReadItemsRequestWithId(id).Build();

var request = ItemSSCRequestBuilder.ReadItemsRequestWithPath(path).Build();

var request = ItemSSCRequestBuilder.SitecoreSearchRequest(text).Build();
```

Validating the Item Identification Parameters

The Sitecore Services Client REST service cannot process certain values of the item parameters, for example, it cannot read an item that you do not have access to or locate an item by an incorrectly formatted parameter. To prevent errors, the Sitecore

Mobile SDKvalidates the submitted item parameters before passing them to the Sitecore Services Client service. It performs the validation on the client whenever possible. If the validation fails, the SDK throws an exception.

The following validation rules are applied to item identification parameters:

- Item ID – the string should contain a GUID, which is a 128-bit number separated by hyphens:

  ```
  const string itemId = "3D6658D8-QQQQ-QQQQ-B3E2-D050FABCF4E1";
  ```

  The hyphens are required. If you pass a string that is null, empty, or consists only of whitespace characters, the request builder throws an exception. In case of other errors, the request is sent to the server, which returns an appropriate response or an error.

- Item Path – the absolute path to the item, beginning with a forward slash:

  ```
  const string itemPath = "/sitecore/content/Home";
  ```

  The leading forward slash is required. If you omit the slash, the request builder throws an *ArgumentException*. If you pass a string that is null, empty, or consists only of whitespace characters, the request builder throws an exception.

Specifying the item source

The item identifications parameters such as the item ID or the item path may be not sufficient to identify the item precisely. The content you retrieve based on these parameters may differ depending on the database in which Sitecore stores the item as well as its language and numeric version.

To specify the item source, use the following item request configuration parameters:

- `database` – to specify a database.
- `language` – to specify a language version.
- `version` – to specify a numeric version.

You can configure the item source in the following ways:

- Per request on the client – the item source parameters are defined in the item request. The values from the request are a top priority.
- Per session on the client – if the request doesn't contain the item source parameters, the SDK uses the values from the *ISitecoreSSCSession.DefaultSource* property of the current session.
- On the server – if neither the request nor the session contains the item source parameters, the SDK does not pass any values to the server. In this case, the server uses its own default values configured by the administrator to define the item source.

The item source is an important parameter that affects both item retrieval and caching. Therefore, you should specify the item source explicitly to always control where the content comes from.

However, to ensure that you retrieve the latest version of the item, you can omit the `version` parameter.

The connection endpoint parameters

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

When you start configuring a session, you should use the connection endpoint parameters to define a host, authentication credentials, and target site.

The Sitecore Mobile SDK provides the following connection endpoint parameters:

- Instance URL
- Credentials

The Instance URL parameter

The `Instance URL` parameter defines a session host – so it is the first parameter that you use to invoke a session initializer.

Property	Details
Session builder API	`AnonymousSessionWithHost(string)` `AuthenticatedSessionWithHost(string)`
Sample value	`http://my-site.com:80`
Parameter position	First in a sequence

Usage guidelines:

- This parameter is required for all session types.
- You cannot invoke this parameter several times in a session – multiple invocations do not compile.
- The value must contain at least one symbol besides whitespace characters.
- If you do not specify the URL scheme in the `Instance URL` parameter, the default value is `http`. For example, `my-site.com` is resolved as http://my-site.com.

The Credentials parameter

To restrict the access to your content, you should set up user permissions in the Security Editor and pass the user name and password to an authenticated session using the `Credentials` parameter:

```
var session =
SitecoreSSCSessionBuilder.AuthenticatedSessionWithHost(instanceUrl)
          .Credentials(loginAndPasswordProvider)
    .BuildSession();
```

To build an anonymous session, skip the `Credentials` parameter:

```
var session =
SitecoreSSCSessionBuilder.AnonymousSessionWithHost(instanceUrl)
.BuildSession();
```

Property	Details
Session builder API	`Credentials(IScCredentials)`
Sample value	n/a
Parameter position	Immediately after the `Instance URL` parameter

Usage guidelines:

- This parameter is required for an authenticated session.
- You cannot invoke this parameter several times in a session – multiple invocations do not compile.
- The value must contain at least one symbol besides whitespace characters.

Sitecore Mobile SDK accepts the credentials via the `IScCredentials` interface. You can either use the default password provider or implement a custom credentials provider.

The SDK provides default cross-platform implementation that pass user credentials to web requests. The implementation use the string class to store credentials, so this implementation is not secure, if you would like to secure credentials you should implement a class derived from the IScCredentials interface.

Send feedback about the documentation to docsite@sitecore.net.

The item request configuration parameters

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

Sitecore Mobile SDK includes functionality to create, retrieve, delete, and update items in the Sitecore instance. The request builder syntax is identical for all the types of item requests. However, the scope of applicable methods varies depending on the operation type. This topic outlines the usage guidelines for the item request methods and parameters.

The Sitecore Mobile SDK provides a set of methods to define the following properties:

- Database
- Language
- Version
- Fields
- Paging
- Standard fields
- Sorting

The Database (database) method

To configure the database to get items from, use the `Database()` method.

Property	Details
Mobile SDK 2.0 (SSC-only) method	`Database(string)`
Item Service API parameter	`database`

Usage guidelines:

- This is an optional parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol besides whitespace characters.

The Language (language) method

To define the item language, use the `Language()` method.

Property	Details
Mobile SDK 2.0 (SSC-only) method	`Language(string)`
Item Service API parameter	`language`

Usage guidelines:

- The `language` parameter is not applicable to item deletion requests.
- This is an optional parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol besides whitespace characters.

The Version (version) method

To define the item version, use the `Version()` method. To retrieve the latest version of an item, do not invoke this method.

Property	Details
Mobile SDK 2.0 (SSC-only) method	`Version(int?)`
Item Web API parameter	`version`

Usage guidelines:

- This is an optional parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- You can only use null or integer numbers as the value.
- Pass a positive number to get a specific version of an item.

The AddFieldsToRead (fields) method

To explicitly set the item fields to be retrieved, use the `AddFieldsToRead()` method.

Property	Details
Mobile SDK 2.0 (SSC-only) methods	`AddFieldsToRead(string)` `AddFieldsToRead(IList<string>)`
Item Service API parameter	`fields`

Usage guidelines:

- This is an optional parameter.
- You can invoke this method several times in a request – multiple invocations accumulate values in invocation order.
- The value must contain at least one symbol besides whitespace characters.
- Field names cannot be null or empty.

The ItemsPerPage (perPage) and PageNumber (page) methods

If the request returns a large dataset, for example, a news feed, a photo stream, or search results, loading the whole content may take a long time and cause memory warnings. To avoid this, the Sitecore Item Service API service and the Sitecore Mobile SDK enable you to load data as a series of chunks.

To split a server response into chunks, use the `ItemsPerPage()` and `PageNumber()` methods.

Property	Details
Mobile SDK 2.0 (SSC-only) method	`ItemsPerPage(int)`
Item SErvice API parameter	`pageSize`

Usage guidelines:

- This is an optional parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.

- The value must be a positive number or you get the *ArgumentException* exception.

Property	Details
Mobile SDK 2.0 (SSC-only) method	`PageNumber(int)`
Item Service API parameter	`page`
Value range	0 to TotalItemsCount/ItemsPerPage + 1

Usage guidelines:

- This is an optional parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must be a positive number or zero or you get the *ArgumentException* exception.

You must either specify both parameters or omit both of them, otherwise the code does not compile.

```
var request = ItemSSCRequestBuilder.SitecoreSearchRequest("home")
 .PageNumber(0)
 .ItemsPerPage(2)
 .Build();
var response = await this.session.RunSitecoreSearchAsync(request);
```

The paging options are only available for reading requests because other operations on items do not require partial execution.

The includeStandardTemplateFields Method

To define the default fields set, use the *includeStandardTemplateFields()* method.

If true, the standard template fields are part of the data that is retrieved.

Property	Details
Mobile SDK 2.0 (SSC-only) methods	includeStandardTemplateFields(bool)
ItemService API parameter	includeStandardTemplateFields
Required/optional	Optional, default value is false
Multiple invocations	Multiple invocations are forbidden and cause InvalidOperationException
Value validation	Must contain bool value

The Sorting Method

To sort results for *RunSitecoreSearchAsync* method use the *AddDescendingFieldsToSort()* and *AddAscendingFieldsToSort()* methods.

Property	Details

Mobile SDK 2.0 (SSC-only) methods	AddDescendingFieldsToSort(string) AddDescendingFieldsToSort(IList<string>) AddAscendingFieldsToSort(string) AddAscendingFieldsToSort(IList<string>)
ItemService API parameter	sorting
Required/optional	Optional
Multiple invocations	Multiple invocations accumulate values in invocation order
Value validation	Must contain at least one symbol besides whitespaces Field names cannot be null or empty

The item source parameters

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

Because Sitecore stores items in different databases and languages, you may need to construct a session that requests items in a particular language from a particular database. To do that, use the `DefaultDatabase` and `DefaultLanguage` parameters:

```
var session = SitecoreSSCSessionBuilder.AnonymousSessionWithHost(instanceUrl)
                                       .DefaultDatabase("web")
                                       .DefaultLanguage("en")
                                       .BuildSession();
```

The DefaultDatabase parameter

Property	Details
Session builder API	`DefaultDatabase(string)`
Sample value	`"master"`, `"web"`, `"core"`
Parameter position	Any position after the `Instance URL` and `Credentials` parameters

Usage guidelines:

- This parameter is optional for all session types.
- You cannot invoke this parameter several times in a session – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol besides whitespace characters.

The DefaultLanguage parameter

Property	Details
Session builder API	`DefaultLanguage(string)`
Sample value	`"en"`, `"da"`, `"fr"`, `"ger"`, `"ja"`, `"cn"`
Parameter position	Any position after the `Instance URL` and `Credentials` parameters

Usage guidelines:

- This parameter is optional for all session types.
- You cannot invoke this parameter several times in a session – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol besides whitespace characters.

Because the builder is order insensitive and both the `DefaultDatabase` and the `DefaultLanguage` parameters are optional, you can specify either both of them, or one of them, or none, for example:

```
var session = SitecoreSSCSessionBuilder.AnonymousSessionWithHost(instanceUrl)
                                       .DefaultDatabase("web")
                                       .BuildSession();
```

Or:

```
var session = SitecoreSSCSessionBuilder.AnonymousSessionWithHost(instanceUrl)
                                       .DefaultLanguage("en")
                                       .BuildSession();
```

Or:

```
var session = SitecoreSSCSessionBuilder.AnonymousSessionWithHost(instanceUrl)
                                       .BuildSession();
```

You can override the default source settings by resetting them in the request object.

The media library configuration parameters

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

You can use the Sitecore Mobile SDK to download files from the media library of a Sitecore instance.

The Media Library Root parameter

To request a media item, the API can use either the full item path or a media path that is relative to the media library root folder. To specify the root folder, use the `MediaLibraryRoot` parameter.

Property	Details
Session builder API	`MediaLibraryRoot(string)`
Sample value	`"/sitecore/media library"`

Parameter position	Any position after the `Instance URL` and `Credentials` parameters

Usage guidelines:

- This parameter is optional for all session types.
- You cannot invoke this parameter several times in a session – multiple invocations cause *InvalidOperationException*.
- The value must begin with a forward slash.

At the time of session initialization, you can change the root folder of the media library that is specified on the server:

```
SitecoreSSCSessionBuilder.AnonymousSessionWithHost(“http://my-host.com”)
                         .MediaLibraryRoot("/sitecore/media library")
                         .BuildSession();
```

Send feedback about the documentation to docsite@sitecore.net.

The session objects and session parameters

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

To send Sitecore Services Client requests, you need a session object. The Sitecore Mobile SDK only provides a public interface for the session. To create a session object, use the `SitecoreSSCSessionBuilder` class API.

Session objects vary in permission types:

Permission type	Session objects
Security permissions	• Anonymous session • Authenticated session
Operation permissions	• ReadOnly session • ReadWrite session

A session object stores the following session configuration parameters:

Parameter type	Session configuration parameters
The connection endpoint parameters	• Instance URL • Credentials •
The item source parameters	• Default database • Default language
The Media Library configuration parameters	• Media Library root item •

You can pass the parameters one at a time in any order. For example:

```
var session =
SitecoreSSCSessionBuilder.AuthenticatedSessionWithHost(“http://my-host.com”)
                         .Credentials(loginAndPasswordProvider)
```

```
                .DefaultDatabase("web")
                .DefaultLanguage("en")
                .MediaLibraryRoot("/sitecore/media library")
                .BuildSession();
```

Building sessions

Once you have set all the session configuration parameters, you can invoke a builder method to create the session.

The Sitecore Mobile SDK provides the following builder methods:

- BuildReadonlySession()

  Use the `BuildReadonlySession()` method to create a read-only session to display the content stored in Sitecore without providing the ability to modify it. When you use this builder method, you must not invoke methods that change the content, or your code does not compile.

  ```
  SitecoreSSCSessionBuilder.AnonymousSessionWithHost("http://my-host.com")
                  .BuildReadonlySession();
  ```

- BuildSession()

  If users need to send data back to the Sitecore instance, use the `BuildSession()` method to create a standard session to allow both read and write access.

  ```
  SitecoreSSCSessionBuilder.AnonymousSessionWithHost("http://my-host.com")
                  .BuildSession();
  ```

Send feedback about the documentation to docsite@sitecore.net.

Creating entities using the Mobile SDK

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

You can create new entities using the RESTful API that the Mobile SDK makes available.

You create a new entity by passing a request that specifies the entity parameters to the `ISitecoreSSCSession.CreateEntityAsync(string id)` method. The following is an example of an entity creation request:

```
var request = EntitySSCRequestBuilder.CreateEntityRequest("1")
               .AddFieldsRawValuesByNameToSet("Title", "some title")
               .AddFieldsRawValuesByNameToSet("Url", null)
               .Build();
ScCreateEntityResponse response = await session.CreateEntityAsync(request);
```

When you construct a request to create an entity, you must specify the following:

- Entity ID
- Appropriate field and field values
- Entity route (optional)

Set entity field values

To submit key-value pairs for the entity fields, use one of the following approaches:

- Submit values one at a time. You can invoke this method several times in a request, multiple invocations accumulate values:

  ```
  var request = EntitySSCRequestBuilder.CreateEntityRequest("1")
  ```

```
            .AddFieldsRawValuesByNameToSet("Title", "some title")
            .AddFieldsRawValuesByNameToSet("Url", null)
            .Build();
```

- Append a dictionary that contains the values:

```
// setup fields beforehand
var fields = new Dictionary<string, string>();
fields.Add("Title", "some title");
fields.Add("Url", null);
var request = EntitySSCRequestBuilder.CreateEntityRequest("1")
                  .AddFieldsRawValuesByNameToSet(fields)
                  .Build();
```

Response processing

As a response, you receive a `ScCreateEntityResponse` object that contains the entity you created:

```
ScCreateEntityResponse response = await session.CreateEntityAsync(request);
String entityId = response.CreatedEntity.Id;
string entityTitle = response.CreatedEntity["Title"].RawValue;
```

You can check the HTTP response code for errors using code like this:

```
int httpStatusCode = response.StatusCode;
```

Note

For an HTTP request, use the following as a guideline:

```
Method: POST
RequestUri: http://host.com/sitecore/api/ssc/aggregate/admin/Todo
RequestBody: {"Title":"Sample title","Url":null,"Id":"1"}
```


Creating items using the Mobile SDK

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

The Sitecore Mobile SDK lets you create new Sitecore items via the API.

To create a new item, you must pass a request that specifies item parameters to the `ISitecoreSSCSession.CreateItemAsync()` method.

A sample item creation request:

```
var request = ItemSSCRequestBuilder.CreateItemRequestWithParentPath("item-id-GUID")
  .ItemTemplateId("item-id-GUID")
  .ItemName(“Name of new item”)
  .Build();
var createResponse = await session.CreateItemAsync(request);
```

When you construct a request to create a Sitecore item, you must:

- Set the item location.
- Set the item name.
- Set the item template.
- (Optionally) Set the item content.

You must specify the following parameters to create an item:

- Item name
- Item template ID
- Parent item ID

Optionally, you can also specify the following item parameters:

- Database
- Language
- Item fields

Setting the item location

To create a new item in Sitecore, you must specify its location in the content tree.

To set the item location, define the parent item by path:

```
var request =
ItemSSCRequestBuilder.CreateItemRequestWithParentParentPath("/path/to/parent")
                     .ItemTemplatePath(“GUID”)
                     .ItemName(“Name of new item”)
                     .Build();
```

In addition to the parent item, you can specify the low-level storage parameters, such as, the item language and the database to store the item in.

Setting the item name

In order for the content of items to be accessible, each Sitecore item must have a name.

To specify the name of the item, use the `ItemName()` method.

Property	Details
Mobile SDK 2.0 (SSC-only) method	`ItemName(string)`
Item Service API parameter	`ItemName (The request body)`

Usage guidelines:

- This is a required parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol besides whitespace characters. You cannot use the following symbols in the item name:
  - A period “.”
  - A dash “-“
  - A slash “/”

  To use the item name in both queries and XPATH expressions, you must conform to the value validation rules. Using forbidden symbols in the item name causes an exception, for example:

  ```
  ItemSSCRequestBuilder.CreateItemRequestWithParentPath (parentPath)
                       .ItemName(“/item/sub-path”) // throws ArgumentException
  ```

The item name must be unique within a given folder. If you specify an existing name, the Item Web API increments the item name automatically.

Setting the item template

The item template defines the list of available fields of the item.

To set the template that you want new items to be based on, specify the template ID.

Specifying the template ID

To identify the item template by its ID, use the `ItemTemplateId()` method.

Property	Details
SSC SDK method	`ItemTemplateId(string)`
Item Service API parameter	`TemplateID (The request body)`

Usage guidelines:

- This is a required parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol enclosed in braces.

A sample request:

```
var request = ItemSSCRequestBuilder.CreateItemRequestWithParentPath(parentItemPath)
                                   .ItemTemplateId(“{GUID-ITEM-ID}”)
                                   .ItemName(“Name of new item”)
                                   .Build();
```

Setting the item content

To compile a list of field names and raw values for the item that you create, use the `AddFieldsRawValuesByNameToSet()` method in the item creation request.

Property	Usage guidelines
Mobile SDK 2.0 (SSC-only) method	`AddFieldsRawValuesByNameToSet(string key, string rawValue)` `AddFieldsRawValuesByNameToSet(IDictionary)`
Item Web API parameter	The request body

Usage guidelines:

- This is an optional parameter.
- You can invoke this method several times in a request – multiple invocations accumulate values.
- You cannot submit duplicate values.
- Both the key and the value must contain at least one symbol besides whitespace characters.

To submit key-value pairs for the item fields, use one of the following approaches:

- Submit values one at a time:

  ```
  var request = ItemSSCRequestBuilder.CreateItemRequestWithParentPath(parentItemPath)
  ```

```
                    .ItemTemplateId(“GUID”)
                    .ItemName(“new item”)
                    .AddFieldsRawValuesByNameToSet("Title", “Device”)
                    .AddFieldsRawValuesByNameToSet("Text", “Smartphone”)
                    .Build();
```

- Append a dictionary that contains the values:

```
// setup fields beforehand
var fields = new Dictionary<string, string>();
fields.Add(“Title”, “Device”);
fields.Add(“Text”, “Smartphone”);
//
var request = ItemSSCRequestBuilder.CreateItemRequestWithParentPath(parentItemPath)
                    .ItemTemplateId(“GUID”)
                    .ItemName(“new item”)
                    .AddFieldsRawValuesByNameToSet(fields)
                    .Build();
```


Delete an entity using the Mobile SDK

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

The Sitecore Mobile SDK lets you delete entities via the API.

To delete an entity:

1. Create a request similar to this:

```
var request = EntitySSCRequestBuilder.DeleteEntityRequest("1").Build();
```

2. Pass the request to the session:

```
ScDeleteEntityResponse response = await session.DeleteEntityAsync(request);
```

3. Use the API to check the request response, for example, to see if the request was successful. For example, use the *Deleted* property to check if the above request was successful:

```
ScDeleteEntityResponse response = await session.DeleteEntityAsync(request);
if(response.Deleted){
  // success
} else {
  //something went wrong
}
```

Note

You can delete an entity with an HTTP request similar to this:

```
Method: DELETE
RequestUri: http://host.com/sitecore/api/ssc/aggregate/admin/Todo('1')
RequestBody: -
```

Delete an item using the Mobile SDK

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

The Sitecore Mobile SDK lets you delete items in the Sitecore content tree from a .NET application via the API.

1. To delete an item, use this request interface: `IDeleteItemsByIdRequest` – identify the item by ID.
2. If you want to remove the item from a particular database, use the `database` parameter.

   The following request removes all the versions of the item in all the languages from the Master database:

   ```
   var request = ItemSSCRequestBuilder.DeleteItemRequestWithId(itemId)
                                      .Database(“master”)
                                      .Build();
   ```

3. Pass the request to the session:

   ```
   var request = ItemSSCRequestBuilder.DeleteItemRequestWithId(itemId)
                                     .Database(“master”)
                                     .Build();
   ScItemsResponse response = await session.DeleteItemAsync(request);
   ```


Retrieve an entity using the Mobile SDK

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

The Sitecore Mobile SDK lets you retrieve entities via the API.

To retrieve an entity:

1. Create a request:
   - To retrieve a single entity by ID, create a request similar to this:

     ```
     var request = EntitySSCRequestBuilder.ReadEntityRequestById("1").Build();
     ```

   - To retrieve all entities, create a request similar to this:

     ```
     var request = EntitySSCRequestBuilder.ReadEntitiesRequest().Build();
     ```

2. Pass the request to the session:

   ```
   ScEntityResponse response = await session.ReadEntityAsync(request);
   ```

3. Use the API to check the request response, for example, to see if the request was successful.

   As a result, you receive an *ScEntityResponse* object that contains one, or several, entities:

   ```
   ScEntityResponse response = await session.ReadEntityAsync(request);
   string entityId = response.CreatedEntity.Id;
   string entityTitle = response.CreatedEntity["Title"].RawValue;
   ```

4. Use the API to check the request response, for example, to see if the request was successful:

   ```
   int count = response.Count(); //entities count in response
   ISitecoreEntity firstEntity = response[0]; //first entity in response
   ```

```
string firstEntityTitle = firstEntity["Title"].RawValue; //the Title field value
```

Note

Use an HTTP request similar to this to retrieve an entity:

```
Method: GET

RequestUri: http://host.com/sitecore/api/ssc/aggregate/admin/Todo
```

To retrieve an entity by ID with an HTTP request:

```
Method: GET

RequestUri: http://host.com/sitecore/api/ssc/aggregate/admin/Todo('1')
```


Retrieving media content using the Mobile SDK

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

To retrieve the media content using the Sitecore Mobile SDK, you should construct and pass a request to a corresponding session building method.

```
// constructing a request

string mediaPath = "/Images/green_mineraly1";

var request = ItemSSCRequestBuilder.DownloadResourceRequestWithMediaPath(mediaPath).Build();

// processing the request

using ( Stream response = await this.session.DownloadMediaResourceAsync(request) )

{

  // working with the media resource stream

}
```

Classes that process images differ between platforms, for example:

Platform	Class name
iOS	`UIImage`
Android	`Bitmap`
Windows Phone	`BitmapImage`

Also, the requested media content may be not an image. For these reasons, the user receives the media content stream after executing the request.

Note

To avoid memory and resource leaks, you should wrap both the stream and the image class in a `using` block.

Setting Media Path

When you request a media resource, the first parameter that you should pass to the request builder is the resource location.

The request builder accepts the following media paths:

- The path relative to the Media Library root folder, which is the raw value of the Image field:

  ```
  "/Images/green_mineraly1"
  ```

- The absolute path of a media item in the Content Editor:

  ```
  "/Sitecore/Media Library/Images/green_mineraly1"
  ```

- The media URL fragment copied from a web browser:

  ```
  "~/media/Images/green_mineraly1.ashx"
  ```

  This fragment must begin with the media hook that has been specified at the time of session initialization.

You can use an absolute path or a media URL fragment for prototyping. However, the safest approach is to use a relative media path.

Searching items using the Mobile SDK

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

To search for an item, pass a request that specifies search parameters to the `ISitecoreSSCSession.RunSearchAsync()` method:

```
var request = ItemSSCRequestBuilder.SitecoreSearchRequest("home").Build();
```

You can use the `Database()` and `Language()` options to specify the item source.

Stored search

You can use a Sitecore stored search. A stored search is a search request that is stored in a Sitecore item (a "search definition item").

The search definition item contains default values for the root item for the search, the template type, and soon. You pass the ID of the search item:

```
var request = ItemSSCRequestBuilder.StoredSearchRequest("search definition item id")
            .Term("home")
            .Build();
```

Stored query

You can use a Sitecore stored query. A stored query is a query request that is stored in a Sitecore item (a "query definition item").

Pass the ID of the query definition item:

```
var request = ItemSSCRequestBuilder.StoredQueryRequest("query definition item id").Build();
```

To run the stored query request, call the `ISitecoreSSCSession.RunStoredQueryAsync()` method, and pass the `StoredQueryRequest`.

The EntityService parameters

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

This topic describes the following parameters of the `EntityService`:

- Entity route parameters
- Custom HTTP parameters

You use the `EntityService` to create, retrieve, update, and delete Sitecore entities.

Entity route parameters

Sitecore uses the entity route to automatically route requests to the appropriate action method. To specify the entity route, you set the following parameters:

- *namespace*
- *controller*
- *id (optional)*
- *action*

You can specify different entity routes for each request like this:

```
var request = EntitySSCRequestBuilder.ReadEntityRequestById(<entityId>)
                                     .Namespace(<namespace>)
                                     .Controller(<controller>)
                                     .TaskId(<id>)
                                     .Action(<action>)
                                     .Build();
```

You specify a common route for the whole session like this:

```
var result = SitecoreSSCSessionBuilder.AuthenticatedSessionWithHost(this.instanceUrl)
                              .Namespace(<namespace>)
                              .Controller(<controller>)
                              .TaskId(<id>)
                              .Action(<action>)
                              .BuildSession();
```

This is per session on the client. If the request does not contain entity route parameters, Sitecore uses the values from the appropriate properties in the current session.

The final URL looks like this:

>

Custom HTTP parameters

You add custom HTTP parameters to the final HTTP request in one of the following ways:

- Submit values one at a time. You can invoke this method several times in a request as multiple invocations accumulate values:

  ```
  .AddParametersRawValues ("paramName", "paramValue")
  ```

  For example:

  ```
  var request = EntitySSCRequestBuilder.ReadEntitiesRequest()
  .AddParametersRawValues ("paramName", "paramValue").Build();
  ```

- Append a dictionary that contains the values:

  ```
  // setup fields beforehand
  var parameters = new Dictionary<string, string>();
  ```

```
parameters.Add("param1", "param1Value");
parameters.Add("param2", "param2Value");
var request = EntitySSCRequestBuilder.ReadEntitiesRequest()
.AddParametersRawValues(parameters).Build();
```

Note

To get entities with a custom HTTP parameter request, follow this example:

```
Method: GET

RequestUri: http://host.com/sitecore/api/ssc/aggregate/admin/Todo? param1Name= param1Value&
param2Name= param2Value

RequestBody: -
```


Update an entity using the Mobile SDK

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

The Sitecore Mobile SDK lets you update entities using the API.

To update an entity, use one of these approaches:

- Create and use a request like this:

  ```
  var request = EntitySSCRequestBuilder.UpdateEntityRequest("1")
  .AddFieldsRawValuesByNameToSet("Title", "New title")
  .Build();
  ```

- Use the `AddFieldsRawValuesByNameToSet()` method with a request to update fields:

  ```
  var fields = new Dictionary<string, string>();
  fields.Add("Title", "new Title");
  fields.Add("Text", "some text");
  var request = EntitySSCRequestBuilder.UpdateItemRequestWithId("item-id-GUID")
  .AddFieldsRawValuesByNameToSet(fields)
  .AddFieldsRawValuesByNameToSet("Manufacturer", "Unknown")
  .Build();
  ```

You can check that the request was successful with the `Updated` property:

```
ScUpdateEntityResponse response = await session.UpdateEntityAsync(request);
if (response.Updated) {
// success
} else {
//something went wrong
}
```

Note

Use an HTTP request similar to this to update an entity :

```
Method: PUT

RequestUri: http://host.com/sitecore/api/ssc/aggregate/admin/Todo('1')

RequestBody: {"Title":"New title"}
```

Send feedback about the documentation to docsite@sitecore.net.

Update an item using the Mobile SDK

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

The Sitecore Mobile SDK includes the content editing API that lets you send new raw values for items to the Sitecore instance and edit the content in different databases and languages.

To update an item:

1. Identify the item that you want to update using the following request interfaces:
   - `IUpdateItemByIdRequest` – to identify the item by ID.
2. If you want the update to affect a particular language, version, or database, specify the item source:
   - To update the item in a particular database, use the `database` parameter.

```
var request = ItemSSCRequestBuilder.UpdateItemRequestWithId("item-id-GUID")
                                   .Database("master")
                                   .Build();
```

   - To pass a new value to a particular language version of the item, specify the `language` parameter.

```
var request = ItemSSCRequestBuilder.UpdateItemRequestWithId("item-id-GUID")
                                   .Language("en")
                                   .AddFieldsRawValuesByNameToSet("Text", "Hello!")
                                   .Build();
```

```
var request = ItemSSCRequestBuilder.UpdateItemRequestWithId("item-id-GUID")
                                   .Language("de")
                                   .AddFieldsRawValuesByNameToSet("Text", "Guten tag")
                                   .Build();
```

If you do not specify a language, the default language version is updated with the new value.

To update a specific version of the item, use the `version` parameter.

```
var request = ItemSSCRequestBuilder.UpdateItemRequestWithId("item-id-GUID")
                                   .Language("en")
                                   .Version(1)
                                   .AddFieldsRawValuesByNameToSet("Text", "Hello!")
                                   .Build();
```

If you do not specify a version, the latest version of the item is updated with the new value.

1. Submit the new field values, using the `AddFieldsRawValuesByNameToSet()` method.

```
var fields = new Dictionary<string, string>();
fields.Add("Title", "Device");
fields.Add("Text", "Smartphone");
var request = ItemSSCRequestBuilder.UpdateItemRequestWithId("item-id-GUID")
```

```
        .AddFieldsRawValuesByNameToSet(fields)
        .AddFieldsRawValuesByNameToSet("Manufacturer", "Unknown")
        .Build();
```

Send feedback about the documentation to docsite@sitecore.net.

Exceptions and error handling

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

Understanding how and when the Mobile SDK throws exceptions is important when building high-quality applications.

This topic outlines some of the typical errors in the Mobile SDK and their causes:

Request construction errors

When you create a request, if you get a validation error, the Item Web API request will not be constructed and you cannot execute it.

The most common causes of request validation errors are:

- Passing an unexpected `null` parameter causes *ArgumentNullException*.
- Passing invalid structures, empty strings, or strings that do not contain anything except whitespace characters causes *ArgumentException*.
- Setting a write-once parameter for the second time causes *InvalidOperationException*.

These exceptions are not wrapped and have no inner exceptions to inspect.

Request execution errors

In the Mobile SDK, the messages generated by the .NET framework are grouped according to the request processing flow, which helps you create informative alert messages and provides the necessary debugging information.

The SDK provides the following set of predefined exceptions:

- *ProcessUserRequestException* – unable to build an HTTP request from the user's request.
- *RsaHandshakeException* – unable to encrypt the user data.
- *LoadDataFromNetworkException* – TCP or HTTP protocol physical connection errors.
- *ParserException* – unable to handle server response properly.
- *WebApiJsonErrorException* – a valid error response returned by the server.

The `SitecoreMobileSdkException` class is a common base class of all the exceptions.

The following sample highlights the exceptions that your application should handle and shows how to order them correctly:

```
try
{
  var request = ItemWebApiRequestBuilder.ReadItemsRequestWithPath(this.ItemPathField.Text)
     .Payload(this.currentPayloadType)
     .AddFieldsToRead(this.fieldNameTextField.Text)
     .Build();
  ScItemsResponse response = await session.ReadItemAsync(request);
// Process response
}
```

```
catch(ProcessUserRequestException)
{
// HTTP request construction failed
}
catch(RsaHandshakeException)
{
// Unable to encrypt user's data
}
catch(LoadDataFromNetworkException)
{
// Connection error
}
catch(ParserException)
{
// Server response is not valid
}
catch(WebApiJsonErrorException)
{
// Server has returned a valid response that contains an error
}
catch(SitecoreMobileSdkException)
{
// Some Item Web API response processing error has occurred.
// This code should not be reached in this example.
}
catch(Exception)
{
// Some exception has been thrown.
// This code should not be reached in this example.
}
```

You can access the original error object in the `Exception.InnerException` property, and the error stack trace in the `Exception.StackTrace` property.

```
try
{
  ScApiSession session = this.instanceSettings.GetSession();
  var request = ItemWebApiRequestBuilder.ReadItemsRequestWithPath(this.ItemPathField.Text)
     .Payload(this.currentPayloadType)
     .AddFieldsToRead(this.fieldNameTextField.Text)
     .Build();
     ScItemsResponse response = await session.ReadItemAsync(request);
// Process response
}
catch(Exception e)
{
```

```
  Debug.WriteLine(e.Message);
  Exception originalError =  e.InnerException;
  Debug.WriteLine(originalError .Message); // access original error
  Debug.WriteLine(originalError .StackTrace);
}
```

To avoid having redundant code, you should catch only `SitecoreMobileSdkException` and handle it in a separate class using the `Type.Equals()` method.

```
try
{
  ScApiSession session = this.instanceSettings.GetSession();
  var request = ItemWebApiRequestBuilder.ReadItemsRequestWithPath(this.ItemPathField.Text)
     .Payload(this.currentPayloadType)
     .AddFieldsToRead(this.fieldNameTextField.Text)
     .Build();
     ScItemsResponse response = await session.ReadItemAsync(request);
// Process response
}
catch(SitecoreMobileSdkException ex)
{
  this.MyErrorProcessor.ShowAlertForSitecoreMobileSdkError(ex);
}
catch(Exception ex)
{
  this.MyErrorProcessor.ShowAlertForUnexpectedError(ex);
}
```

Note

The Sitecore Mobile SDK does not provide error classes that show alerts because they may vary depending on the target platform and may require customization.

Install the Mobile SDK from the Xamarin Component Store

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

This topic describes how to include the *Mobile SDK 2.0 (SSC-only)* component from the Xamarin Component Store in a project using Xamarin Studio.

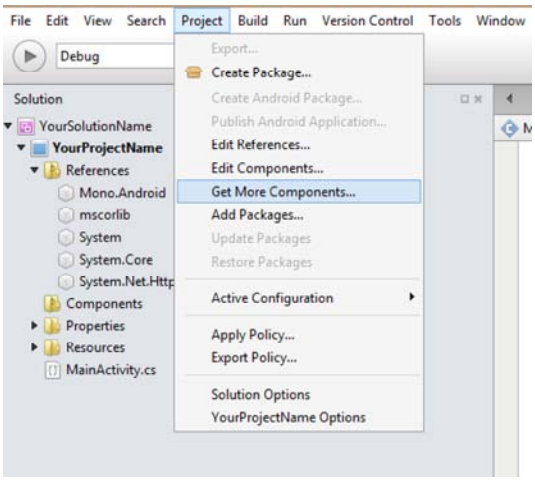
The instructions apply to both Android and iOS platform.

Note

To install components from the Xamarin Component Store, you must first connect Xamarin Studio or Visual Studio to your Xamarin account.

To install the Sitecore Mobile SDK from the Xamarin Component Store in Xamarin Studio:

1. On the toolbar, click Project and then click Get More Components:



2. In the All Components dialog box, in the search field, enter the "`Sitecore Mobile SDK 2.0 (SSC-only) for Xamarin`".
3. In the search results, select "`Sitecore Mobile SDK 2.0 (SSC-only) for Xamarin`" and click Add to App.

When the package is downloaded, you can use the Sitecore Mobile SDK in your project.

Send feedback about the documentation to docsite@sitecore.net.

Requirements for the Sitecore Mobile SDK

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

This topic lists the supported platforms and outlines the client and server system requirements to install and use the Sitecore Mobile SDK.

Supported platforms

You can use the Mobile SDK to develop Sitecore client applications for any of these supported platforms:

- Xamarin iOS
- Xamarin Android
- Windows 8.1 or later
- Windows Phone 8.1 or later

Client Requirements

Before installing the Mobile SDK, you must have the following applications installed and running:

- .NET framework 4.5 or later
- NuGet 2.8.5 or later
- Visual Studio 2015 or later

For iOS and Android application development using the Xamarin platform, you also need:

- Xamarin Studio 6.0 or later installed

Note

Sitecore does not supply any form of license for Xamarin. For more information about the Xamarin platform and licensing, visit xamarin.com.

Server Requirements

To install and use the Sitecore Mobile SDK, your server must meet the following requirements:

- Sitecore Experience Platform 8.1 or later.

Send feedback about the documentation to docsite@sitecore.net.

Resource management for the Mobile SDK

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

Because of memory and CPU limitations, mobile applications must use resources efficiently. This topic describes the resource management techniques that you can apply to optimize your resource allocation when using the Sitecore Mobile SDK:

Resource management for Sitecore Mobile SDK classes

Most classes and interfaces in the SDK are pure, managed objects. However, the session holds references to native resources, for example, a reference to the `HttpClient` instance, which uses a native socket. The credentials provider can access a native, platform-specific keychain.

To properly release unmanaged resources, the session implements the `IDisposable` interface, which requires that:

- If you use the session as a local variable, you must declare and instantiate it in a `using` statement, which calls the `Dispose` method in the correct way and causes the session object to go out of scope as soon as the method is called:

```
using  (var session =
    SitecoreSSCSessionBuilder.AnonymousSessionWithHost(instanceUrl)
                                .DefaultDatabase("web")
                                .DefaultLanguage("en")
                                .MediaLibraryRoot("/sitecore/media library")
                                .MediaPrefix("~/media/")
                                .DefaultMediaResourceExtension("ashx")
                                .BuildReadonlySession())
{
    // Send some requests
}
```

- If you store the session as an instance variable of your class, you must implement the `IDisposable` interface and dispose the session object explicitly:

```
void ReleaseResources()
{
  if (null != this.session)
  {
    this.session.Dispose();
    this.session = null;
  }
}
```

```
public virtual void Dispose()
{
  this.ReleaseResources();
  GC.SuppressFinalize(this);
}
~MyClassThatUsesSSCSession()
{
}
```

To ensure that your application disposes `HttpClient` and credentials properly, wrap both the session and credentials in a `using` block. Otherwise, the application may crash with a memory warning or with the *Too many open files* exception.

The session owns the credentials provided on the initialization and attempts to make a copy of them. If your implementation of the password provider does not support copying, do not wrap it in a `using` block.

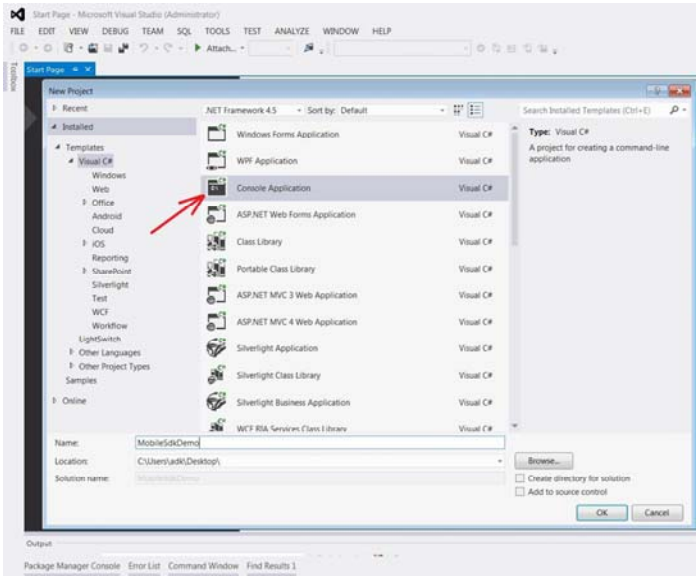
Use the Mobile SDK to build a console application in Visual Studio

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

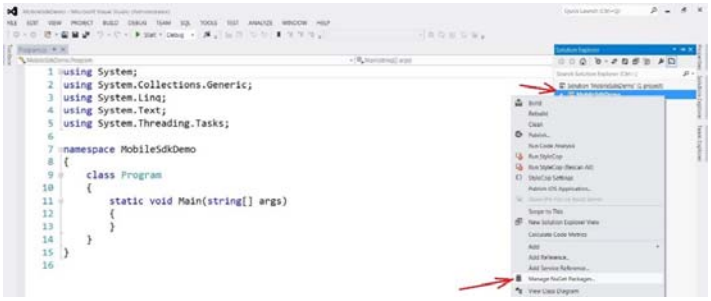
Before you can start using the Sitecore Mobile SDK, you must add its assembly reference to a dependent project, either in your application or in the class library. This topic describes the NuGet-based setup for Microsoft Visual Studio.

To build a console application using the Sitecore Mobile SDK in Microsoft Visual Studio:

1. Create a project in Visual Studio. You can create a desktop application, mobile application and others. This example describes how to create a console application.

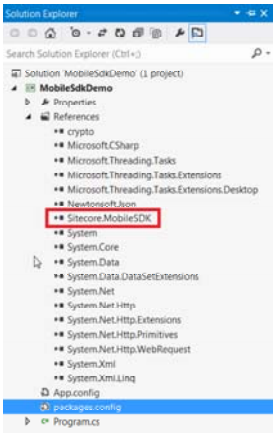


2. To install NuGet packages, in the Solution Explorer window, right-click the project node and click Manage NuGet Packages.



1. In the Manage NuGet Packages dialog box, in the search field, enter `Sitecore.MobileSDK`. (not case-sensitive).
2. In the search results, select Sitecore Mobile SDK 2.0 (SSC-only) for Xamarin and click Install. Accept the license agreements when prompted.

   When the NuGet package is installed, you can see the updated project references in the Solution Explorer.



3. In the Visual Studio Code Editor, add the following code to the `Main()` function of your application:

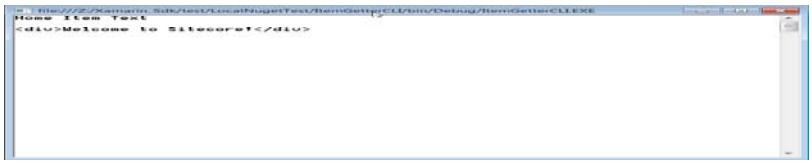
```
private static void Main(string[] args)
{
  string instanceUrl = "http://my.site.com";
  using (var demoCredentials = new ScUnsecuredCredentialsProvider("username", "pass", "domain"))
  using
  (
    var session =
      SitecoreSSCSessionBuilder.AuthenticatedSessionWithHost(instanceUrl)
                               .Credentials(demoCredentials)
                               .DefaultDatabase("web")
                               .DefaultLanguage("en")
                               .MediaLibraryRoot("/sitecore/media library")
                               .MediaPrefix("~/media/")
                               .DefaultMediaResourceExtension("ashx")
                               .BuildReadonlySession())
  {
    var request =
      ItemSSCRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
                           .Build();
    var readHomeItemTask = session.ReadItemAsync(request);
    // cannot use "await" in main
```

```
        Task.WaitAll(readHomeItemTask);
        ScItemsResponse items = readHomeItemTask.Result;
        string fieldText = items[0]["Text"].RawValue;
    Console.Clear();
        Console.WriteLine("Home Item Text");
        Console.WriteLine();
        Console.WriteLine(fieldText);
        Console.ReadKey();
      }
  }
```

4. Add the following namespaces to your project to ensure that the code compiles properly:

```
using System;
using System.IO;
using System.Threading.Tasks;
using Sitecore.MobileSDK.API;
using Sitecore.MobileSDK.API.Items;
using Sitecore.MobileSDK.API.MediaItem;
using Sitecore.MobileSDK.API.Request.Parameters;
using Sitecore.MobileSDK.PasswordProvider
```

A sample output of the application:



Send feedback about the documentation to docsite@sitecore.net.

Use the Mobile SDK to build an Android application in Visual Studio

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

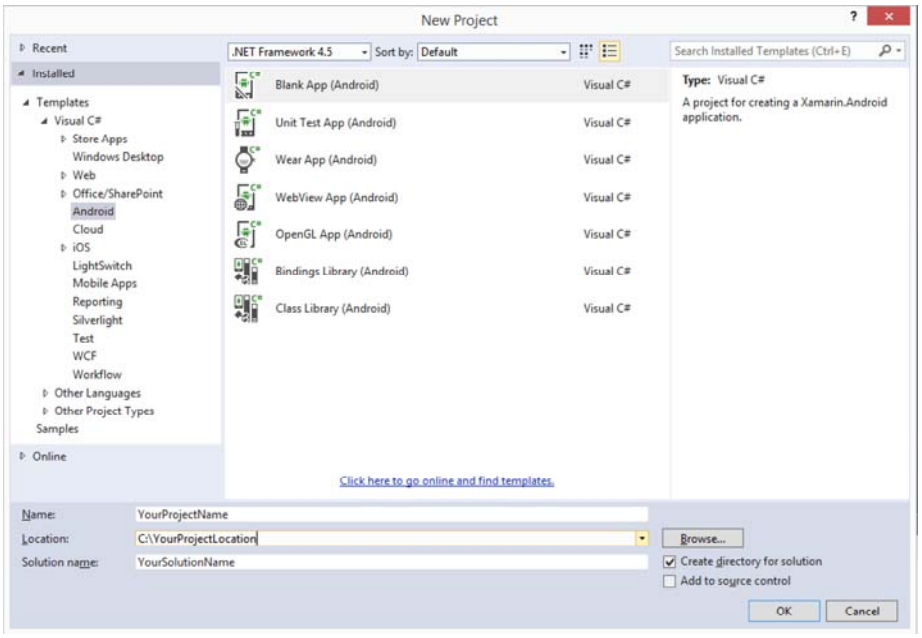
Before you can start using the Sitecore Mobile SDK, you must add its assembly reference to a dependent project. This topic describes how to install the Sitecore Mobile SDK using NuGet and create an Android application in Microsoft Visual Studio.

Note

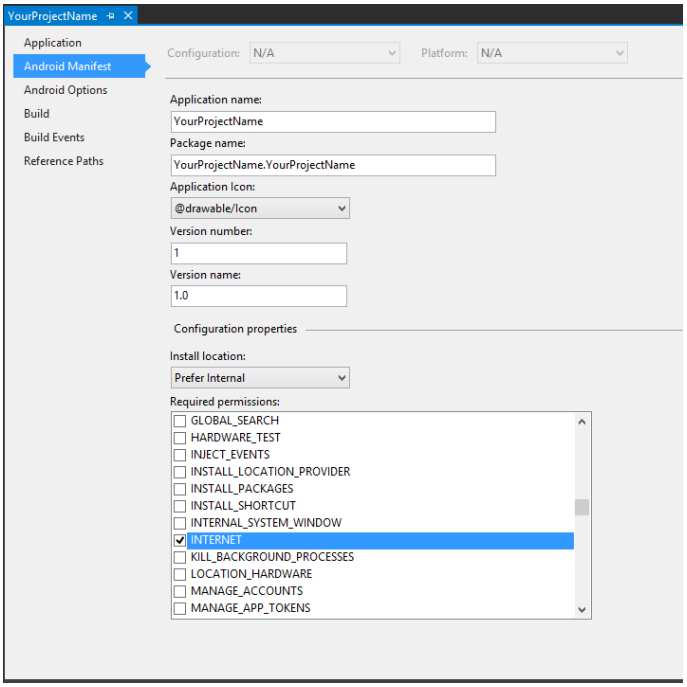
Ensure that you have also installed Xamarin Studio and that you have activated the installation with your Xamarin account.

To build an Android application using the Sitecore Mobile SDK in Microsoft Visual Studio:

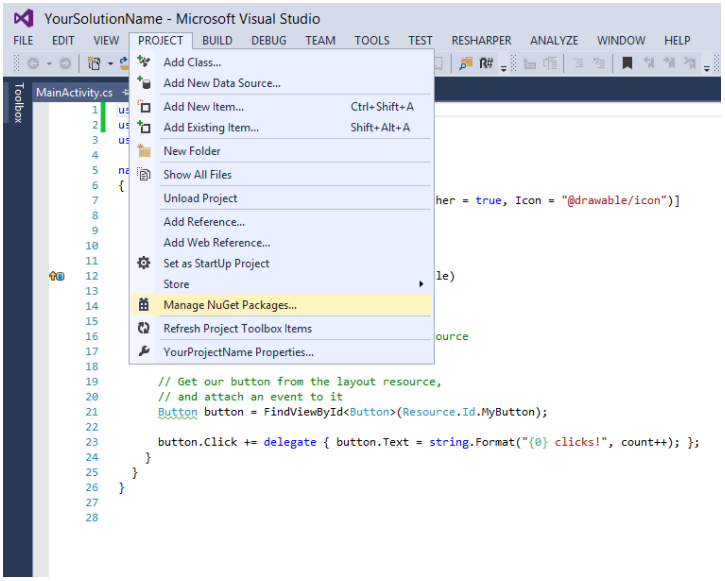
1. To create a project in Visual Studio, in the New Project wizard, click Visual C#, Android, Blank App (Android).



2. To add the permission for accessing network resources to the Android manifest file, in the Solution Explorer, right-click the project and select Properties.
3. In the Project Designer window, click Android Manifest, and in the Required permissions list box, select Internet.

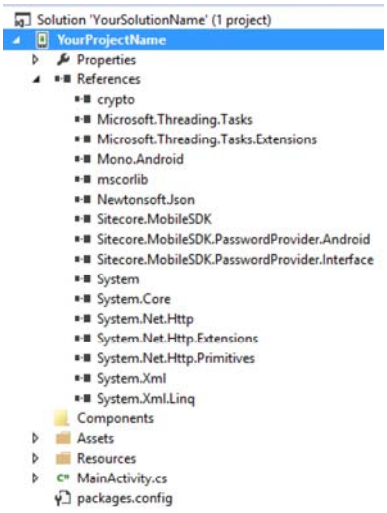


4. To add the Sitecore Mobile SDK packages to your solution, on the toolbar, click Project and then click Manage NuGet Packages.



5. In the Manage NuGet Packages dialog box, in the search field, enter `Sitecore.MobileSDK` (not case-sensitive).
6. In the search results, select Sitecore Mobile SDK 2.0 (SSC-only) for Xamarin and click Install.

7. When the packages are added, you can see them in the *References* folder of the project in the Solution Explorer.

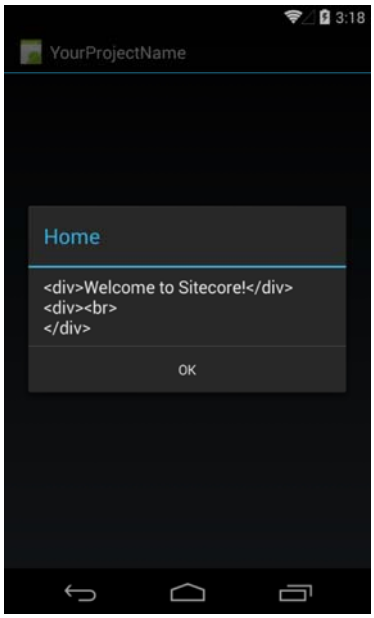


Now you can use the Sitecore Mobile SDK. To build a sample application, add the following lines to the `MainActivity.cs` file:

```
namespace YourProjectName
{
  using Android.App;
  using Android.Content;
  using Android.OS;
  using Sitecore.MobileSDK.API;
  using Sitecore.MobileSDK.API.Items;
  using Sitecore.MobileSDK.PasswordProvider
  using Sitecore.MobileSDK.API.Request.Parameters;
  [Activity(Label = "YourProjectName", MainLauncher = true, Icon = "@drawable/icon")]
  public class MainActivity : Activity
  {
    protected async override void OnCreate(Bundle bundle)
```

```
    {
      base.OnCreate(bundle);
      string instanceUrl = "http://my.site.com";
  using (var demoCredentials = new ScUnsecuredCredentialsProvider("username", "pass", "domain"))
  using
  (
    var session =
      SitecoreSSCSessionBuilder.AuthenticatedSessionWithHost(instanceUrl)
                               .Credentials(demoCredentials)
                               .DefaultDatabase("web")
                               .DefaultLanguage("en")
                               .MediaLibraryRoot("/sitecore/media library")
                               .MediaPrefix("~/media/")
                               .DefaultMediaResourceExtension("ashx")
                               .BuildReadonlySession())
  {
    var request =
      ItemSSCRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
                           .Build();
    var readHomeItemTask = session.ReadItemAsync(request);
    // cannot use "await" in main
    Task.WaitAll(readHomeItemTask);
    ScItemsResponse items = readHomeItemTask.Result;
    string fieldText = items[0]["Text"].RawValue;
        var dialogBuilder = new AlertDialog.Builder(this);
        dialogBuilder.SetTitle(items[0].DisplayName);
        dialogBuilder.SetMessage(fieldText);
        dialogBuilder.SetPositiveButton("OK", (object sender, DialogClickEventArgs e) => { });
        dialogBuilder.Create().Show();
      }
    }
  }
}
```

When it launches, the application displays an alert with the corresponding item name and field value.



Note

If you get this error:

Deployment failed because the device does not support the package's minimum Android version. You can change the minimum Android version in the Android Application section of the Project Options.

Open the Project Designer using the Properties command on the Project menu, and on the Application tab, change the *Minimum Android to target* setting to *Android 4.0.*

Send feedback about the documentation to docsite@sitecore.net.

Use the Mobile SDK to build an Android application in Xamarin Studio

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

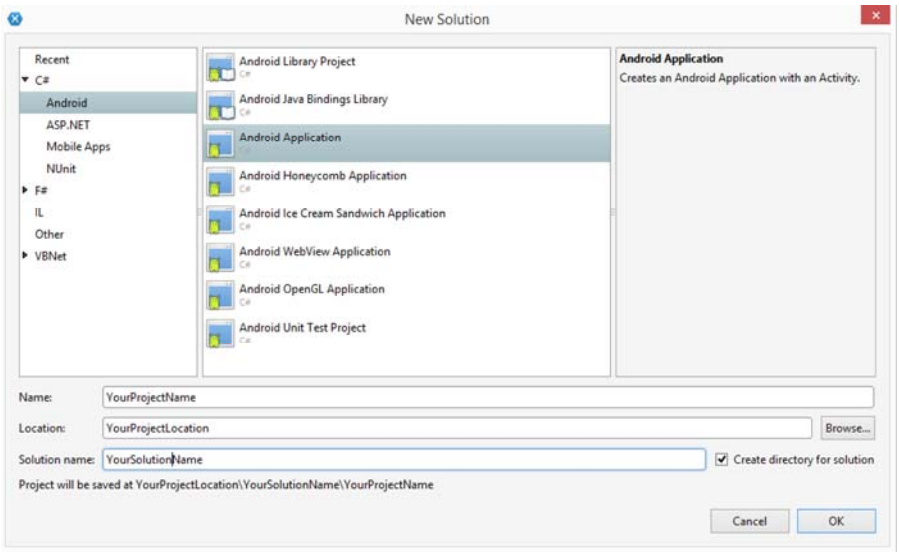
Before you can start using the Sitecore Mobile SDK, you must add its assembly reference to a dependent project. This topic describes how to install the Sitecore Mobile SDK using NuGet and create an Android application in Xamarin Studio.

Note

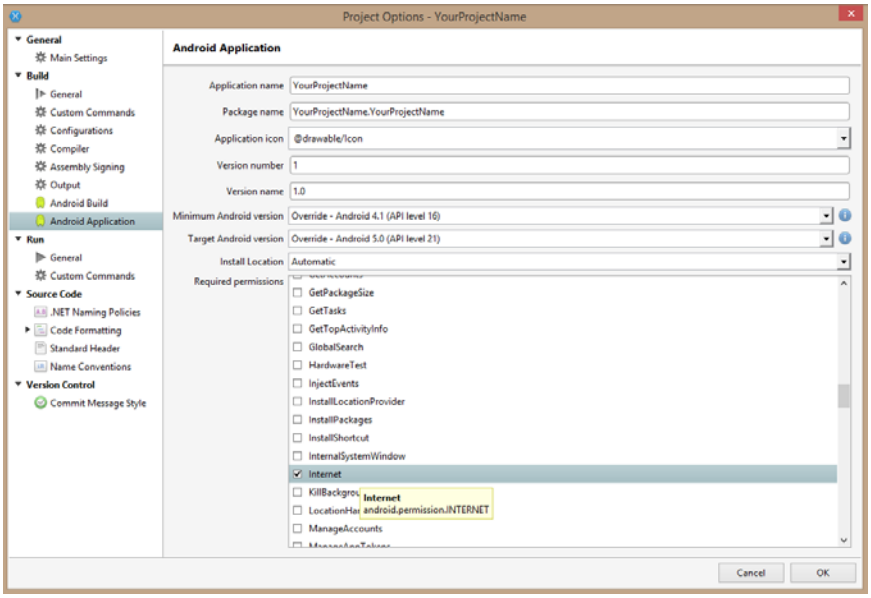
Ensure that you have installed Xamarin Studio and activated the installation with your Xamarin account.

To build an Android application using the SDK in Xamarin Studio:

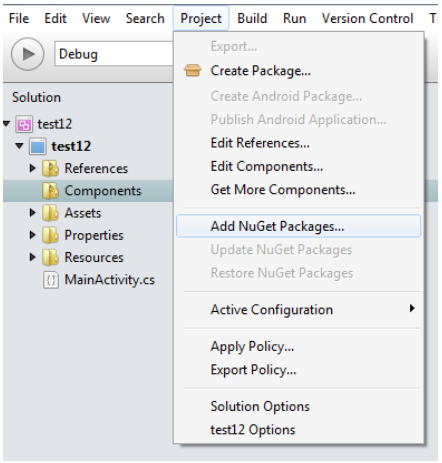
1. To create a project in Xamarin Studio, in the New Solution wizard, click C#, Android, Android Application.



2. To add the permission for accessing network resources to the Android manifest file, in the Solution Pad, right-click the project and select Options.
3. In the Project Options window, click Android Application, and in the Required permissions list box, select Internet.

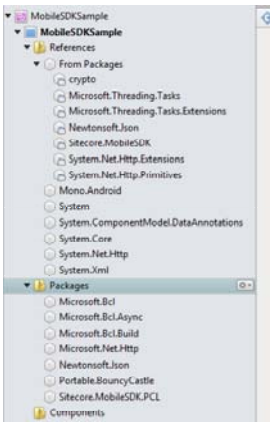


4. To add the Sitecore Mobile SDK packages to your solution, on the toolbar, click Project and then click Add NuGet Packages.



5. In the Add Packages dialog box, in the search field, enter the `Sitecore.MobileSDK` ID. The ID is not case-sensitive.
6. In the search results, select Sitecore Mobile SDK 2.0 (SSC-only) for Xamarin and then click Add Package.

When the packages are added, you can see them in the *Packages* folder of the project in the Solution Pad.

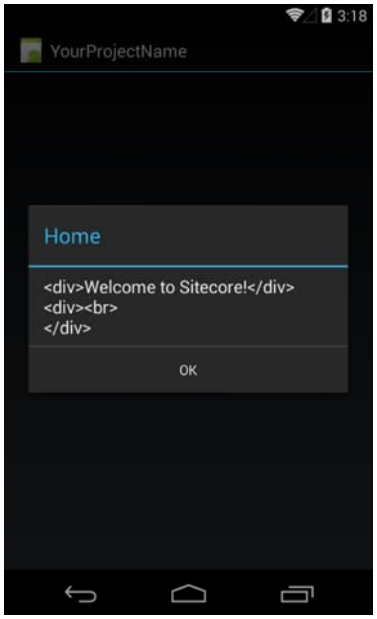


Now you can use the Sitecore Mobile SDK. To build a sample application, use the following code example:

```
using System;
using Android.App;
using Android.Content;
using Android.OS;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Sitecore.MobileSDK.API;
using Sitecore.MobileSDK.API.Items;
using Sitecore.MobileSDK.API.Session;
using Sitecore.MobileSDK.PasswordProvider;
namespace MobileSDKSample
{
  [Activity (Label = "MobileSDKSample", MainLauncher = true, Icon = "@drawable/icon")]
  public class MainActivity : Activity
  {
      protected async override void OnCreate (Bundle bundle)
      {
          base.OnCreate (bundle);
   string instanceUrl = "http://my.site.com";
   using (var credentials = new ScUnsecuredCredentialsProvider("login", "password", "domain"))
   using (
        var session =
          SitecoreSSCSessionBuilder.AuthenticatedSessionWithHost(instanceUrl)
          .Credentials(credentials)
          .DefaultDatabase("web")
          .DefaultLanguage("en")
          .MediaLibraryRoot("/sitecore/media library")
          .MediaPrefix("~/media/")
          .DefaultMediaResourceExtension("ashx")
          .BuildReadonlySession())
```

```
    {
  var request =
  ItemSSCRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
                          .Build();
  ScItemsResponse items = await session.ReadItemAsync(request);
  string fieldContent = items[0]["Text"].RawValue;
    string itemName  = "Home Item Text";
    var dialogBuilder = new AlertDialog.Builder (this);
    dialogBuilder.SetTitle (itemName);
    dialogBuilder.SetMessage (fieldText);
    dialogBuilder.SetPositiveButton ("OK", (object sender, DialogClickEventArgs e) => {});
    dialogBuilder.Create ().Show ();
  }
  }
    }
}
```

When it launches, the application displays an alert with the corresponding item name and field value.



Note

If you get this error:

Deployment failed because the device does not support the package's minimum Android version. You can change the minimum Android version in the Android Application section of the Project Options.

In the `Android.Manifest.xml` file, change the *Minimum Android version* setting to *Android 4.0*.

Use the Mobile SDK to build an iOS application in Xamarin Studio on Mac

This topic describes the Mobile SDK for Sitecore 8.2. This version of the SDK uses Sitecore.Services.Client while earlier versions use the Item Web API. The different versions are generally not compatible.

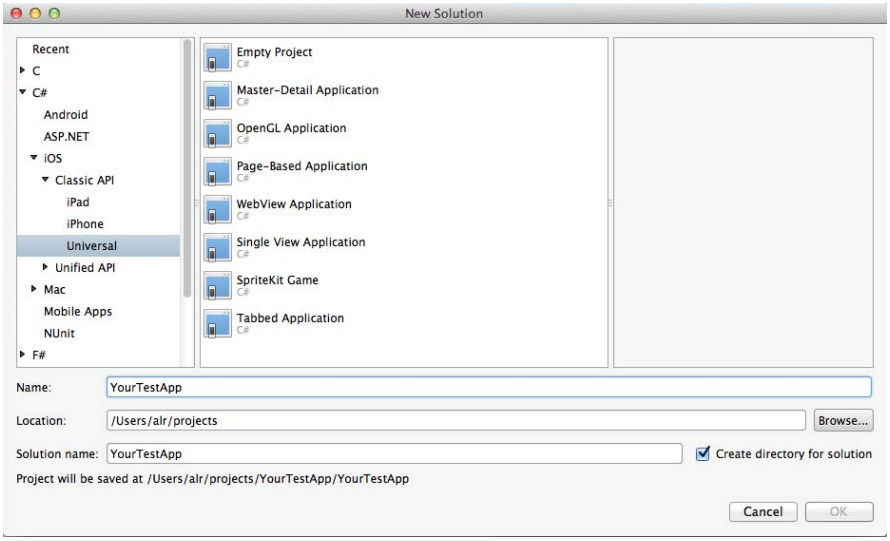
Before you can start using the Sitecore Mobile SDK, you must add its assembly reference to a dependent project. This topic describes how to install the Sitecore Mobile SDK for Xamarin using NuGet and create an iOS application in Xamarin Studio on Mac.

Note

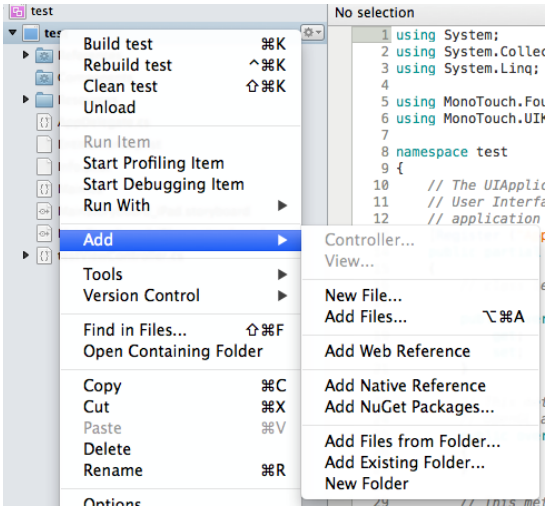
To become familiar with the fundamentals of iOS application development with Xamarin, visit http://developer.xamarin.com/guides/ios/getting_started/hello,_iOS/.

To build an iOS application using the Sitecore Mobile SDK for Xamarin in Xamarin Studio on Mac:

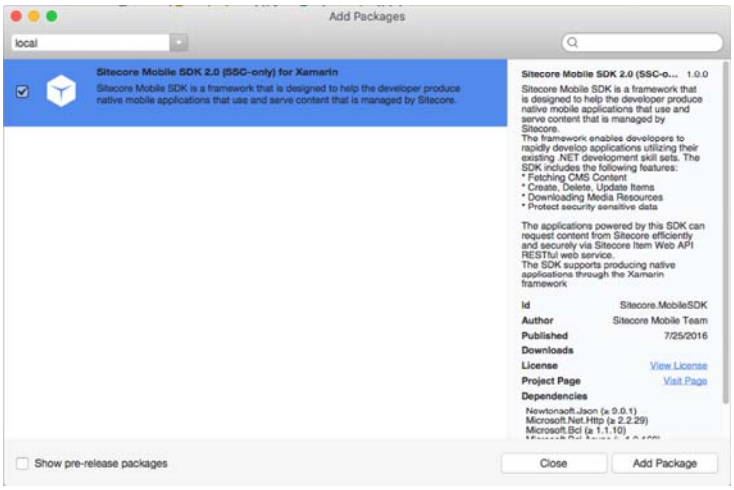
1. To create a project in Xamarin Studio, in the New Solution wizard, click iOS, Classic API or Unified API for a 64 bit application, Universal, Single View Application.



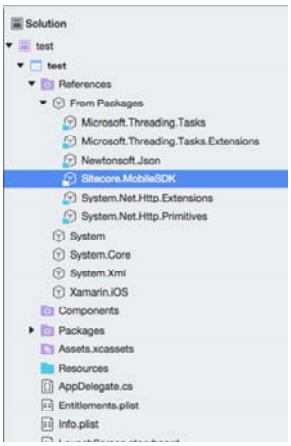
2. To add the Sitecore Mobile SDK packages to your solution, in the Solution Pad, right-click the project node, click Add, Add NuGet Packages.



3. In the Add Packages dialog box, in the search field, enter the `Sitecore.MobileSDK` ID. The ID is not case-sensitive.
4. In the search results, select Sitecore Mobile SDK 2.0 (SSC-only) for Xamarin and click Add Packages.



When the NuGet packages are added, you can see the updated project references.



5. In the editor, add the following code to the `ViewDidAppear()` function of your application:

```
public override async void ViewDidAppear(bool animated){  base.ViewDidAppear(animated);
  string instanceUrl = "http://my.site.com";
using (var demoCredentials = new ScUnsecuredCredentialsProvider ("login", "password", "domain"))
using
(
  var session =
    SitecoreSSCSessionBuilder.AuthenticatedSessionWithHost(instanceUrl)
                             .Credentials(demoCredentials)
                             .DefaultDatabase("web")
                             .DefaultLanguage("en")
                             .MediaLibraryRoot("/sitecore/media library")
                             .MediaPrefix("~/media/")
                             .DefaultMediaResourceExtension("ashx")
                             .BuildReadonlySession())
{
  var request =
    ItemSSCRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
                         .Build();
  ScItemsResponse items = await session.ReadItemAsync(request);
```

```
string fieldContent = items[0]["Text"].RawValue;
  UIAlertView alert = new UIAlertView(
    "Home Item Demo",
    fieldContent,
    null,
    "Ok",
    null);
  alert.Show();
}}
```

6. Add the following namespaces to your project to make the code compile correctly:

```
using System;
using System.Drawing;
using System.Threading.Tasks;
using Foundation;
using UIKit;
using Sitecore.MobileSDK.API;
using Sitecore.MobileSDK.API.Items;
using Sitecore.MobileSDK.API.Request.Parameters;
using Sitecore.MobileSDK.PasswordProvider
```

When it launches, the application displays an alert with the contents of the Text field of the *Home* item.